Carnegie Mellon
Software Engineering Institute

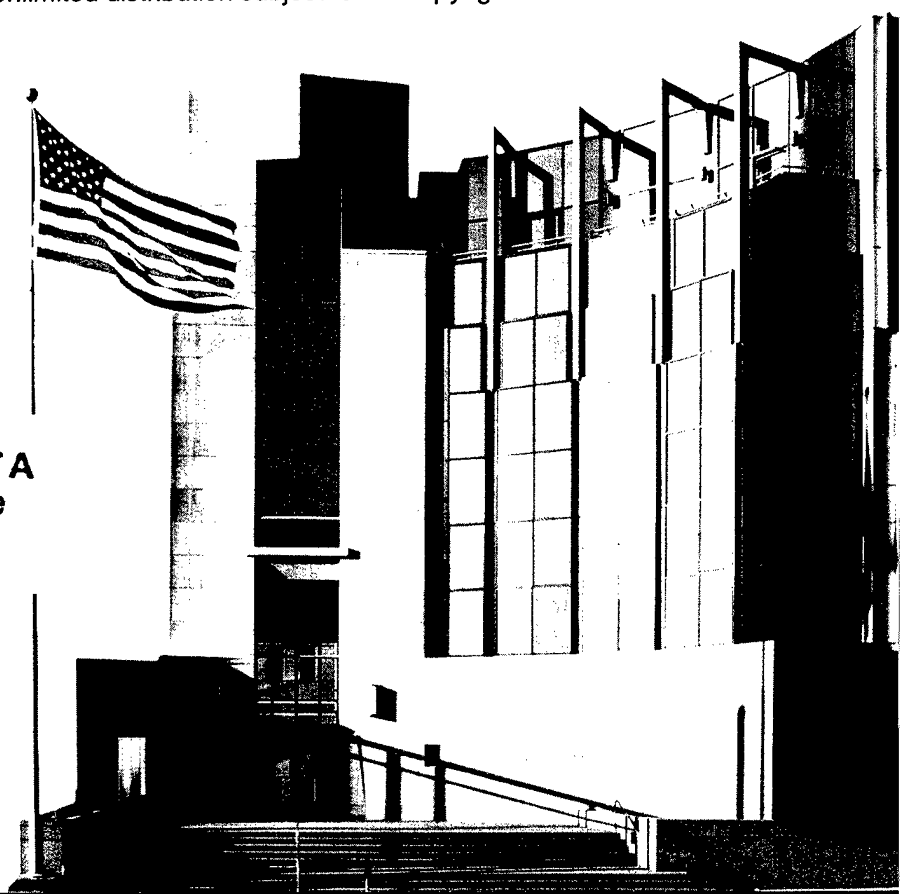
Creating and Using Software Architecture Documentation Using Web-Based Tool Support

Judith A. Stafford

September 2004

Software Architecture Technology Initiative

Technical Note
CMU/SEI-2004-TN-037



Creating and Using Software Architecture Documentation Using Web-Based Tool Support

Judith A. Stafford

September 2004

Software Architecture Technology Initiative

Technical Note
CMU/SEI-2004-TN-037

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

Acknowledgements

The DSA Wizard, which is the inspiration for this report, was produced by Deborah Seok as a master's project at Tufts University. We are grateful to Deborah for bringing many insights to the project and creating a prototype documentation system that enables us to demonstrate the concepts underlying the wizard's design. All screen shots used in this document are from Deborah's product. In addition, we would like to thank Robert Nord and Paul Clements for their valuable feedback on drafts of this report.

Abstract

Documenting software architecture (DSA) is a crucial facet in the development of a software system, yet often it is carried out in a haphazard fashion, if at all. Lack of attention to the documentation results from insufficient guidance about what should be documented and when and how to capture the information so that system stakeholders find it useful. The book *Documenting Software Architectures: Views and Beyond* provides such guidance in the DSA approach, and this report describes the conceptual design for a documentation system based on that approach. A system is envisioned that enables the architect to capture architectural decisions and related artifacts as a living repository that can communicate information to stakeholders who might be both geographically and temporally distributed. The system must communicate in a way that allows each stakeholder quick and easy access to information relevant to the person's role in the software development process. This report describes a design prototype that demonstrates a Web-based approach to creating, communicating, and using software architecture throughout the life of the system.

1 Software Architecture Documentation

The architecture of a software system offers little value if not documented well enough for it to be used to create, maintain, and evolve the system. Despite the importance of architecture documentation, it often is given insufficient attention or overlooked entirely as part of the software development process. The architect often does not understand what should be documented, and no tool support exists to keep the documentation current and enable the architect to communicate information to relevant system stakeholders.

This report describes the design for a documentation system that supports capturing software architecture as a Web-based, database-backed, access-controlled document that performs as a living repository of architectural information throughout the system's life. The design supports the *Views and Beyond* (V&B) approach espoused in the book, *Documenting Software Architectures: Views and Beyond* [Clements 03]. We used a prototype design called the DSA Wizard to exemplify such a system and to demonstrate the creation and use of documentation captured in this way. The system could be instantiated for single use, as we have done with the DSA Wizard, using specialized tools such as Dreamweaver, MySQL, and PHP, or included as an architecture documentation template in a content management system to support multiple uses.

1.1 The Views and Beyond Approach

Documenting software architecture is a matter of describing the appropriate architectural views and then adding information that applies to more than one view. Architectural views represent different software perspectives that support the varying needs of the architecture's stakeholders. In the V&B approach, perspectives are categorized as *viewtypes* (which are specialized by *styles*) that provide the guidance for creating a *view*. These three concepts form the conceptual backbone of the V&B approach and are described in the following paragraphs.

Viewtypes — In the V&B approach, three types of views provide the foundation for structuring architecture documentation: module, component and connector (C&C), and allocation. A viewtype is defined in terms of a set of element types and relation types. For instance, the elements of the module viewtype are modules—principle implementation units of the software. Relation types describe how the modules are related to each other (e.g., *is-part-of*, *depends-on*, *is-a*). Each element and relation type may have associated properties that are also described as part of the viewtype's definition.

Styles — In a viewtype, patterns of interactions often occur that can be captured as *styles*, or patterns if you will, that provide architectural solutions based on quality requirements and other software concerns. Architectural styles may refine element and relation types and, in addition, include a set of constraints on the interactions. For example, the Module Viewtype includes the *Decomposition* style, which focuses on how system responsibilities are spread across the implementation units and how the modules are decomposed into submodules. This style is based on the *is-part-of* relation and includes constraints specifying that the decomposition graph may not contain loops, and no module may be part of more than one module in a view. While the book describes several styles associated with each viewtype, these styles are not intended to be all-encompassing. The system architect remains free to define or adopt styles from other systems when appropriate. When a style is created or adopted, the architect must define it within the architecture documentation.

Views — When the architectural styles are determined, the architect constructs views on the software to support the stakeholders' needs. Views describe some aspect of the system in terms of system-specific elements and relations. Relations are defined in terms of the element and the style's relation types and submit to the constraints imposed by the style.

1.2 Principles of Sound Documentation

The Web-based design supports the V&B approach, including the seven principles of documentation prescribed by the authors of the approach. These principles, described below, provide the foundation for the authors' guidance on documentation. Although these principles are not specific to architecture documentation, they apply in this context as well as they do in others.

Principle 1: Write from the reader's viewpoint.

A document is read only if it meets the needs of, and is usable by, its intended audience. Material written in streams of consciousness or using arcane terminology is unlikely to meet the reader's needs and thus is unlikely to be read or consulted often.

Principle 2: Avoid unnecessary repetition.

While repetition sometimes reinforces a point, its use in technical information becomes troublesome over time. Repetition is the root of inconsistency. Keeping track of all repeats is difficult, if not impossible; thus, repeated information becomes inconsistent over time, and attempts to avoid these inconsistencies are costly.

Principle 3: Avoid ambiguity.

This principle might better be stated as "avoid unintended ambiguity" because software architecture, by its nature, is ambiguous in areas that remain undecided until the system is

implemented. Nevertheless, if a decision is made, the documentation must communicate it unambiguously so that system stakeholders do not misinterpret it. Such misinterpretation can lead to confusion, incorrect implementation, or problems during system verification and validation.

Principle 4: Use a standard organization.

Usually a document is not read more than once, if that. Yet, if it is successful, readers will refer to it numerous times. Providing a standard organization not only helps a reader quickly find information, but also provides the architect with guidance on what needs to be captured and what has or hasn't been captured at any given time.

Principle 5: Record rationale.

The reasoning behind the decisions is just as important as the decisions themselves. Architecture documentation lives with the system and, as most developers have experienced, the reasoning behind a decision may be forgotten in as little as a few weeks. Understanding the rationale behind decisions helps the architect refrain from revisiting decisions, helps designers understand why specific choices were made, and supports system evolution by stating explicitly that certain decisions were based on the context and technological constraints imposed at the time.

Principle 6: Keep documentation current but not too current.

While documentation should not become out-of-date, disseminating recent modifications to certain stakeholders may be ill-advised at times. Documentation remains the final authority, and stakeholders consult it for guidance when making decisions about the system. Including information that might not be final does not help them. Organizations are well-advised to determine a documentation release plan that is appropriate to their practices and processes.

Principle 7: Review documentation for fitness of purpose.

Documentation is successful only if it meets its readers' needs. Thus, these readers are the ones who determine its usefulness and should be encouraged to provide feedback about whether it meets their needs.

We designed the Web-based documentation system specifically to support the documenter in following the principles discussed above.

1.3 Stakeholders

In both Sections 1.1 and 1.2, we refer to the architecture's *stakeholders*. These are the people for whom the documentation is created; therefore, understanding them and their needs is

critically important to creating successful documentation. A list of stakeholders definitely should include software developers, maintainers, managers, analysts, and the architect, but normally will include many other classes of users. Before documentation begins, the architect creates a table of stakeholders versus views in which each stakeholder's needs are recorded. Each view comes with a cost, and each view comes with a benefit. Available resources must be weighed against stakeholders' needs. The architect must balance these opposing concerns carefully. Once these decisions are made, the system design described in this report enables the architect to provide each stakeholder with a documentation view tailored to the person's specific needs.

2 Providing Web-Based Support for Architecture Documentation

An organization can capture and distribute documentation in a variety of forms: printed notebooks, electronic format, or a Web site. Each option is appropriate for a given project depending on its size, distribution of the development team, software processes, and organizational practices. For a small project carried out in a single building, a paper document or simple electronic file suffices. The printed notebook provides a sense of stability for developers of large projects in which the architecture is complete before further development takes place. Digital distribution provides ease of dissemination and can be backed by printed documents. However, when a team follows an iterative or incremental approach to development, the document becomes a living repository of architectural guidance. The architect creates and disseminates the documentation to the appropriate stakeholders when needed. No matter what form a documentation system takes, its standard organization helps a reader quickly find information and guides the architect on which information to capture.

The Web-based, repository-backed documentation system design described in this report especially supports the capture and use of architectural decisions for projects taking an iterative or incremental approach to development. If given sufficient support for configuration management, however, this design also can be used effectively when more rigid processes are employed. Taking a Web-based, repository-backed approach to documentation provides an environment in which software architecture documentation can be captured and used by multiple people, in concert, across geographic and temporal boundaries. At the same time, this approach readily supports access control to give each stakeholder the appropriate access and read/write permissions as determined by organizational policies and/or the architect.

2.1 Guiding Principles

The documentation system design is based on these principles:

- Support the V&B approach to architecture documentation.
- Support the seven principles of sound documentation.
- Provide stakeholders with easy, immediate access to needed documentation regardless of their geographic distribution.

Our Web-based approach easily addresses each of these principles.

2.1.1 Supporting the V&B Approach

We based the prototype forms on the documentation templates in *Documenting Software Architectures: Views and Beyond*, thereby providing a structure to guide the architect in using this approach [Clements 03]. The design includes forms for capturing all information along with guiding tips from the book, which occur at relevant points throughout the document and are tailored to the user's needs.

2.1.2 Supporting the Principles of Sound Documentation

Principle 1: Write from the point of view of the reader.

The Web-based design includes a component to support role-based user login and access control. Within the context of a particular document, once a stakeholder logs in, the system generates a view tailored to that stakeholder's needs and access rights. The view might provide a part of the document as a Web form to the architect and that same part as a Portable Document Format (PDF) file for a developer; a junior architect is likely to benefit from guidance at a level that would cause a senior architect frustration. Clickable pop-up windows show tips relevant to the role of the currently logged-in stakeholder. For instance, if a junior architect is creating a primary presentation, the tip might provide guidance on organizational standards or notations, while the tip for a developer would provide the semantics of the notation used by the architect.

Principle 2: Avoid unnecessary repetition.

The database back end of the system allows each piece of information to be created and maintained as individual files and accessed by multiple users in a variety of contexts. Hyper-linking allows the illusion of redundancy while completely removing its need. A glossary captures definitions in a single place and provides a readily accessible and consistent reference for all stakeholders. Similarly, an acronym list is created and is readily accessible at any time. Diagrams and other files can be created once and referenced from multiple locations throughout the document.

Principle 3: Avoid ambiguity.

While the use of natural language always allows for varied interpretation, the design reinforces the need for precision and clarity. In addition, the use of Unified Modeling Language (UML) and other more formal languages is supported by way of file upload.

Principle 4: Use a standard organization.

A template, by nature, enforces the use of a standard organization. With this system's design, we use Web forms as templates that provide rigid structure to the document's various elements.

Principle 5: Record rationale.

The design provides entries specific to recording rationale in appropriate places and encourages their use by querying the author if this section of the form is not used.

Principle 6: Keep documentation current but not too current.

Keeping the documentation current is simple in Web-based documentation; however, issues associated with "too current" are more pressing in this type of environment. The Web-based documentation system is best backed with a configuration management system that makes available various levels of currency based on the user's role.

Principle 7: Review documentation for fitness of purpose.

The Web-based design supports easy and immediate feedback on both the documentation's form and content as email links on every Web page.

2.1.3 Providing Stakeholders with Easy, Immediate Access to Information

A given stakeholder must be able to find the desired material quickly, with minimum distraction. The material should be presented to the user in a form appropriate to the task at hand. For instance, a manager most likely needs the high-level overview, whereas a developer probably wants quick access to the description of a particular module view. The architect creates architectural documents, whereas the manager only reads them. The chief architect might be in Massachusetts and interested overseers in Washington, DC. When a modification is made to the architecture, the architect might need to communicate that information to developers immediately. Web-based documentation is available, constrained by role-based access management, 24 hours a day around the globe. New documentation can be disseminated immediately to all relevant parties.

2.2 DSA Documentation System Design

Web technologies available today support the production of documentation systems that allow multiple users to create, manage, and use shared files across the globe without concern for geographical, political, or temporal boundaries. Web sites can be backed with a configuration management system, and access rights can be controlled using password

protected login. Users can be assigned roles, and the view of the Web site can be varied according to predefined user needs. These powerful characteristics of Web-based technologies assure a robust platform on which to base a documentation system.

2.2.1 Document Structure

The structure of a documentation system includes the following components:

- general information. This section provides information about software architecture and architecture documentation in general. It might be garnered from published materials or from within the organization.
- forms. Forms, which allow direct input, cut and paste, or file submission depending on the material being captured, follow the structure of the Microsoft Word template that is included with this document as Appendix A.
- tips. Tips, as prominently displayed, clickable pop-ups, appear at relevant locations to help guide the user. Flash diagrams can be used as mini-tutorials within the tips environment. As an example, Figure 1 shows the first two frames of a flash sequence that describes a sequence diagram. The complete flash sequence contains 10 frames, each describing 1 aspect of the sequence diagram.

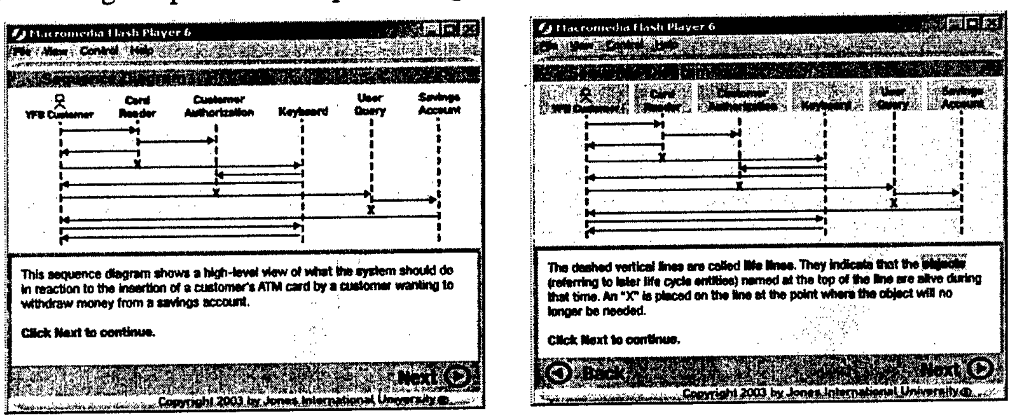


Figure 1: Flash Diagrams Functioning as Mini-Tutorials

- glossary. Each document must have an associated glossary. A glossary link is displayed prominently on each page of the Web site, and entries are individually linkable. Any person with write permission to the document can add links.
- acronym list. Each document must include an acronym list, with a link from each page and from any acronym within the text.

2.2.2 Role-Based Access Management

Each stakeholder uses or references the software architecture documentation from a different point of view and thus requires varying degrees of access to its sections. To support these

needs, the documentation system must include role-based controlled access, which is configured on a system-by-system basis by the architect or other authorized person.

- front page
  - The section available for all persons provides general information about software architecture documentation and the contents of the system itself (e.g., the prologue from *Documenting Software Architectures: Views and Beyond* [Clements 03], a list of software systems that contain architecture documentation, and contact names of responsible persons).
  - The login screen for a returning user includes the user name and password. Successful login produces a menu of software architectures for which the stakeholder is authorized.
  - The link to registration for new users takes them to a page that includes stakeholder roles that can be preassigned to users by the architect or other authority or selected by the user, depending on user rights within a particular organization.
- tips. Each role contains a set of associated tips. Tips' location and content vary by role and are stakeholder dependent. For instance, a senior documenter does not need the same level of support as a junior documenter. A tip attached to the C&C Pipe-and-Filter View form would describe the types of architectural elements that constitute pipes and filters, the purpose of the view, and so forth for a documenter. For a developer, however, it would describe the implementation characteristics and implications of using pipes and filters.

2.2.3 Forms and File Input and Output

Forms support multiple input formats whenever possible, including

- manual entry. Users type information directly into the form, and it is uploaded as an ASCII text file.
- cut and paste. Users retrieve previously typed information from other documents and use the cut and paste mechanism to complete the form, which is then uploaded as an ASCII text file.
- file upload. Users are allowed to upload previously generated documents and diagrams as Multipurpose Internet Mail Extensions (MIME) filetypes.

The output form is stakeholder-role dependent. The following three options are supported and made available based on the user's assigned role. For instance, a documenter may access a view packet as a PDF file as well as a form, but a developer may access it only as a PDF file.

- filled-in forms
- PDF versions of forms
- PDF version of entire document structured according to the Word template provided in Appendix A

2.2.4 Configuration Management System

Configuration management of a living document housed in a Web environment raises complex issues beyond those normally associated with multiple-authored documents. An organization must manage carefully the creation and dissemination of information according to that organization's specific policies. It must assign read and write permissions. In addition, it must stage configuration releases and create various configurations for the different stakeholder roles. A configuration management back end allows an organization to group parts of the document into a variety of configurations at any given time. This capability allows the architect to modify the document as the architecture matures and still provide the document to the developer as staged releases.

2.2.5 System Configuration and Extensibility

Due to variation among software systems and organizations, a documentation system must allow the architect, or another authorized person, to reconfigure and, in some cases, extend the management and documentation support offered by the system.

Configurable items include

- stakeholder types. Stakeholder types with names and associated permissions must be created according to the culture of the organization so that users can understand the meaning and limitations of their roles.
- tips. The language of Tips is likely to vary and should be editable.
- forms. Forms must be editable to allow adding fields and changing labels.

Extensible items include

- tips. Support must be provided for adding tips.
- forms. Addition and deletion of forms must be supported.

3 Example Documentation System

We implemented a prototype of the design called the DSA Wizard. To create the documentation system, we used the Web-site tool Macromedia Dreamweaver; the graphic animation tool Macromedia Flash; the database back end MySQL; and the scripting language PHP. The following subsections provide a tour of the DSA Wizard, highlighting important aspects of the program's design.

3.1 Welcoming the User

At the wizard's Welcome Screen, users are allowed access to the general information via links in the Site Map pane. Information provided includes descriptions of the three viewtypes under the topic "Purpose and Scope of the SAD," as shown in Figure 2. Returning users are invited to log in or change their password; new users can follow a link to the registration page.

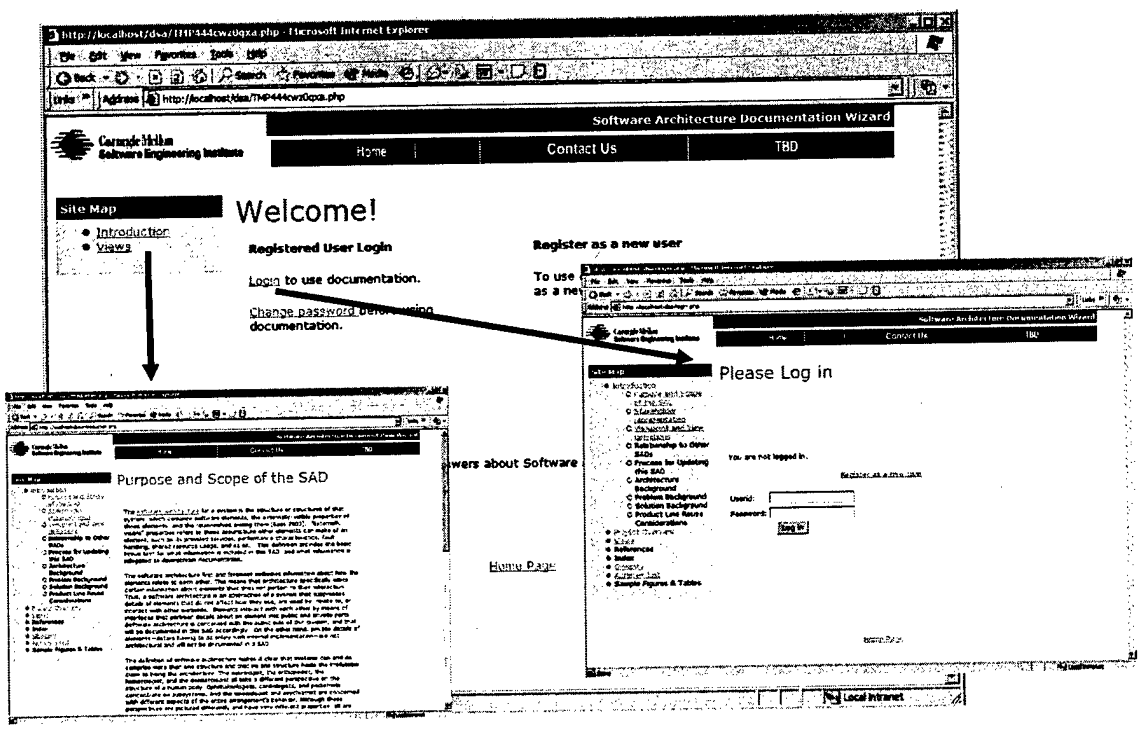


Figure 2: Welcome Page for the DSA Wizard

3.2 Role Assignment and Selection

A principal aspect of the documentation system is role-based view control. Associated with each stakeholder role are document access permissions that are assigned by the architect or other authorized person to each element of stored documentation. When users complete the registration form, the system presents them with a list of available roles. As is the case with the document permissions, the allowed user roles are configured by the architect or other authorized person. In addition to access control, document elements may be offered in varied formats on a role-by-role basis. Each user may have multiple stakeholder-role assignments but must log in using a specific stakeholder role. The registration screen provides the user with a list of stakeholder roles from which to choose, as shown in Figure 3.

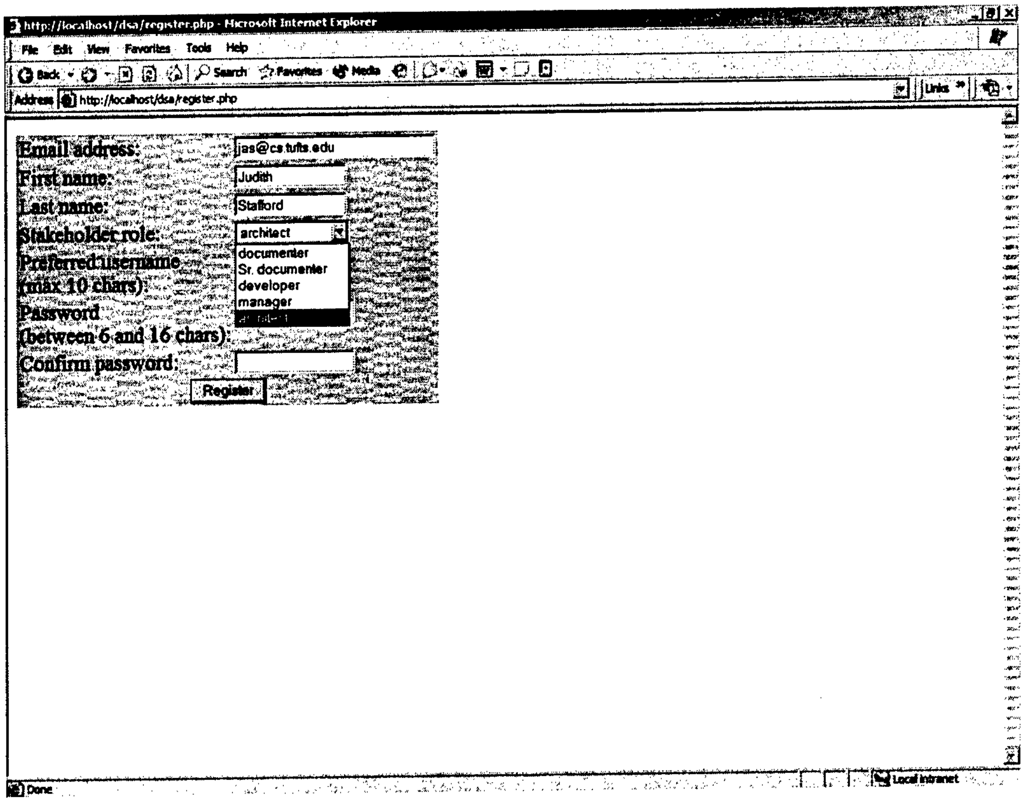


Figure 3: Mandatory Stakeholder-Role Selection as Part of User Registration

This aspect of the login process is configured by the architect or another authorized individual, depending on organizational practices. When a stakeholder logs in with a specific role, it is identified on every screen viewed.

If a document system contains documents for more than one software architecture, user roles may vary from architecture to architecture, and registration must include a document-selection option.

3.3 Varied Site-Map Contents

The system provides document views based on the needs and rights of the stakeholder. Figure 4 shows the architect's site map, which appears once an architect has successfully logged in to the document. It includes a link to instructions for updating the software architecture document.

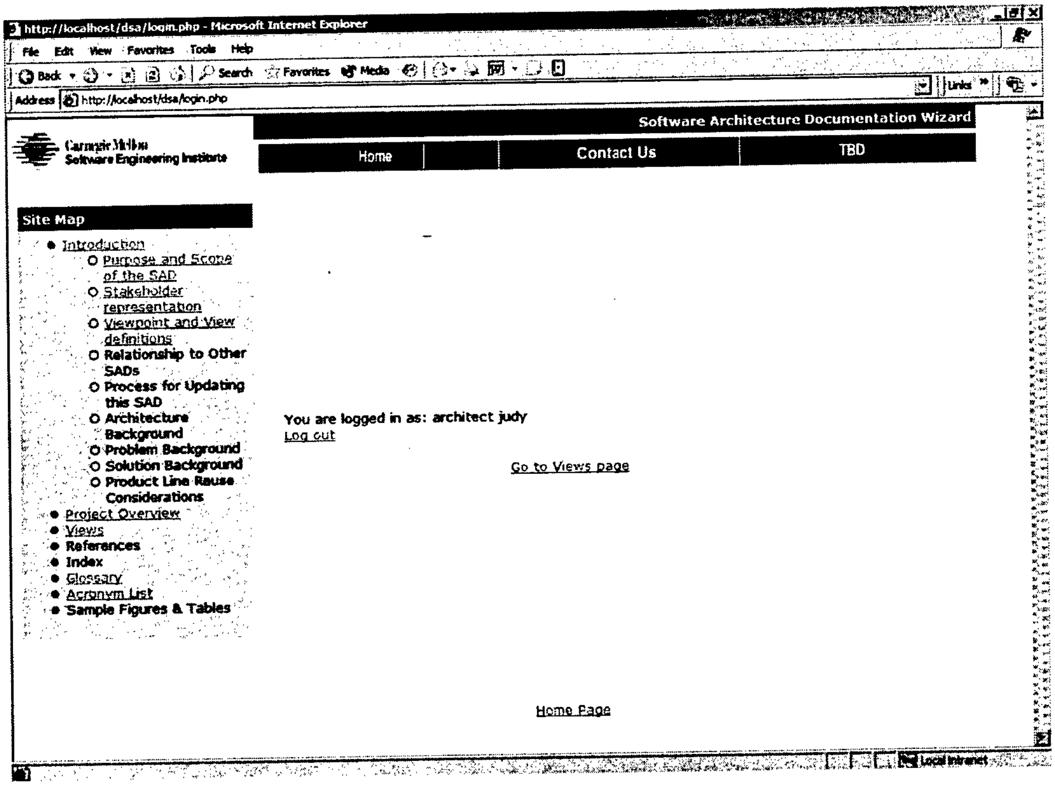


Figure 4: Role-Specific Contents in the Site Map

3.4 Hyperlinked Glossary and Acronym List

Every software architecture document must contain a glossary of terms and an acronym list for easy reference by document readers. A hypertext environment allows the glossary and acronym list to be hyperlinked from within the document, avoiding the confusion and possible inconsistencies that result when definitions are provided in multiple locations for easy access. The glossary and acronym list also can be accessed as a hyperlink from the text itself. For example, as shown in Figure 5, clicking on the term "software architecture" takes the user to a definition of that term. This feature makes understanding complicated definitions or acronyms (that might contain other acronyms) much easier. For instance, interpreting the acronym ECS (EOSDIS Core System) requires knowing that EOSDIS stands for Earth Observing System Data and Information System.

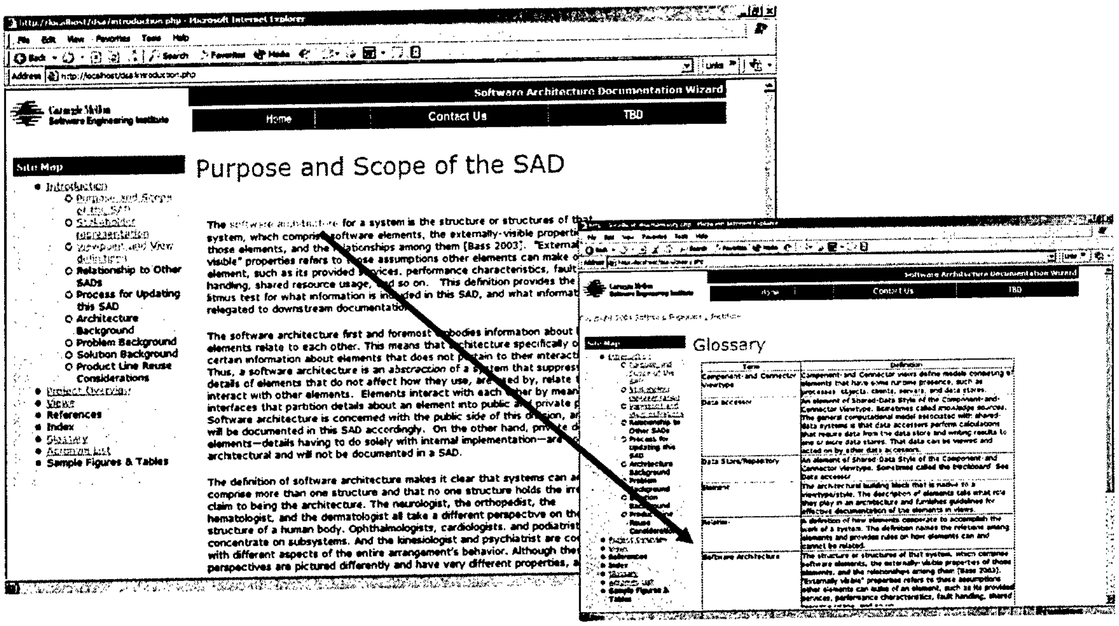


Figure 5: Hyperlinked Mandatory Glossary

3.5 Role-Based Presentation

Not only do available screens vary based on roles, but the form in which they appear can vary also. Figure 6 shows the Project Overview as a Web page for a developer and as a Web form for the documenter; the documenter may enter or revise the form, but the developer may only view it.

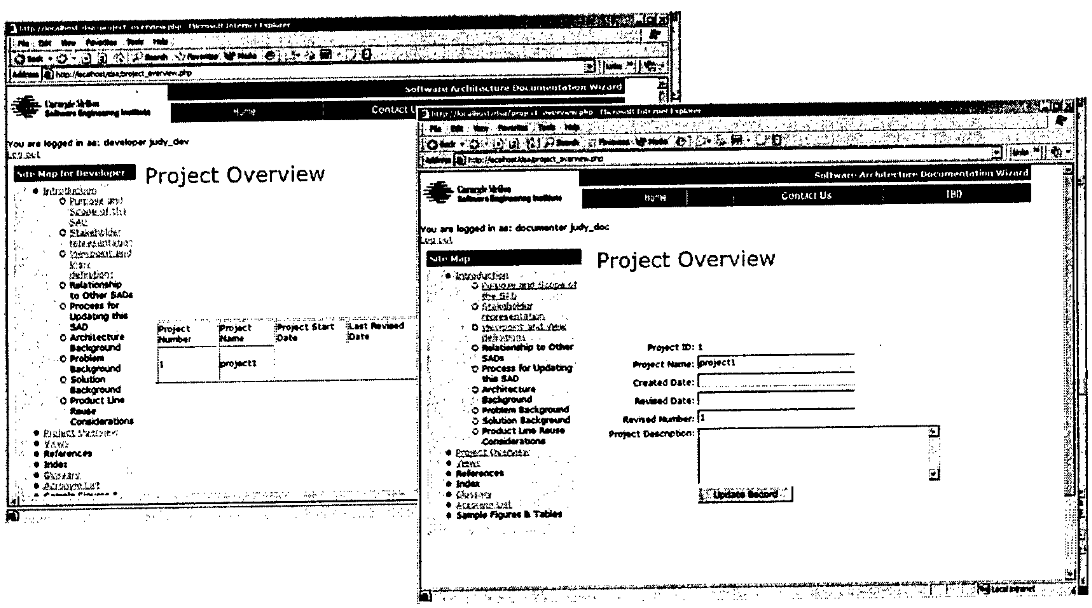


Figure 6: Project Overview Screen as Presented to a Developer and a Documenter

Figure 7 shows the Views screen presentations for a developer on the left and for a documenter on the right. The developer may select from available architectural views, while the documenter is allowed to add a new view or edit an existing view.

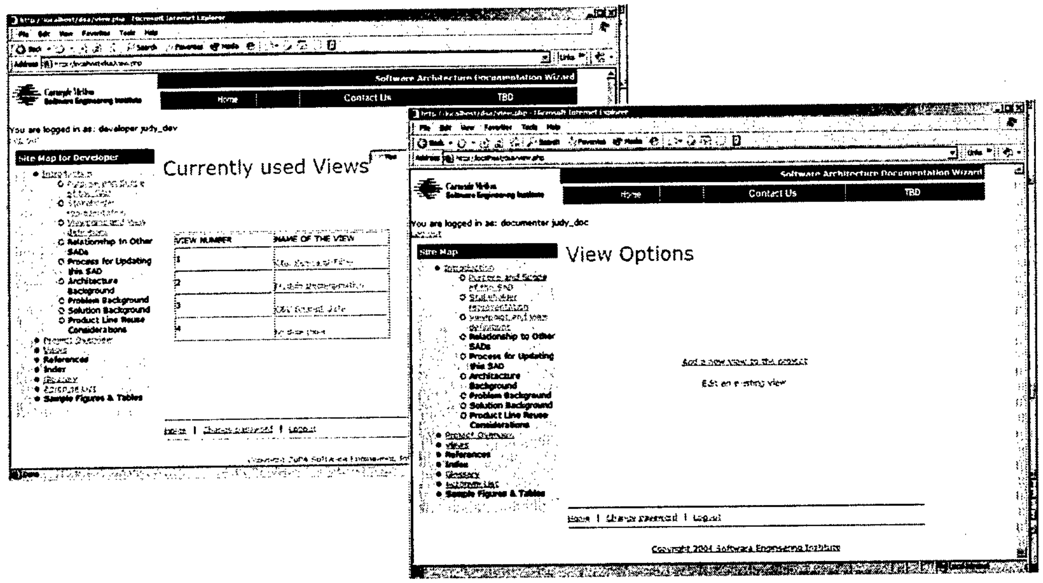


Figure 7: Role-Based Presentations of the Views Screen

The developer's screen presents the view as a PDF document:

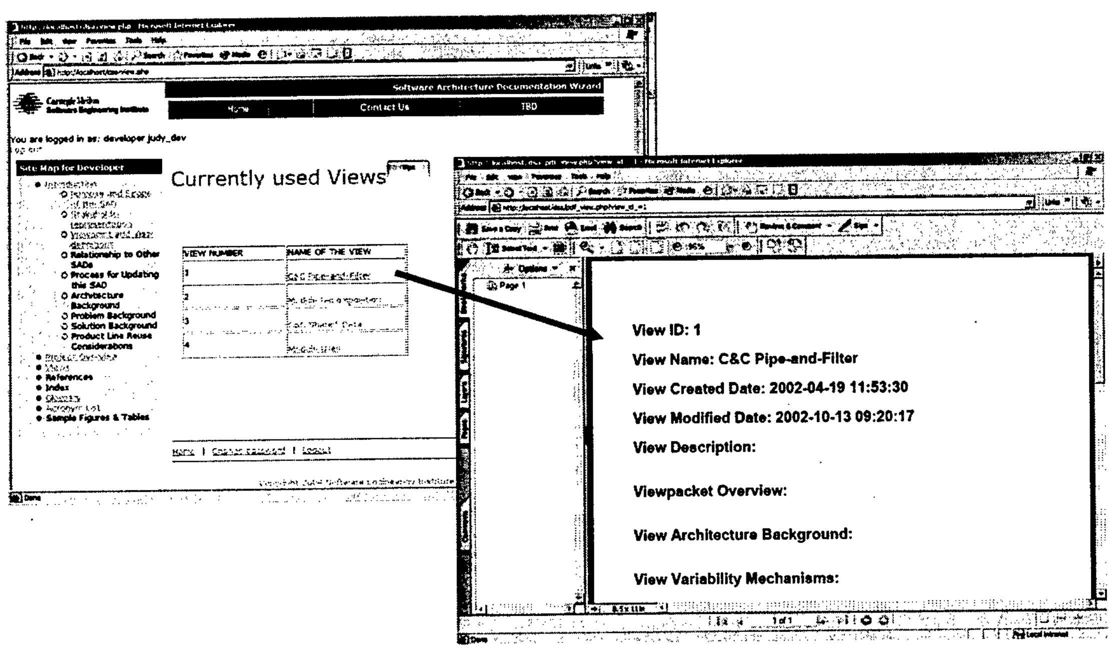


Figure 8: Developer's View

The documenter's screen presents the same view as a form:

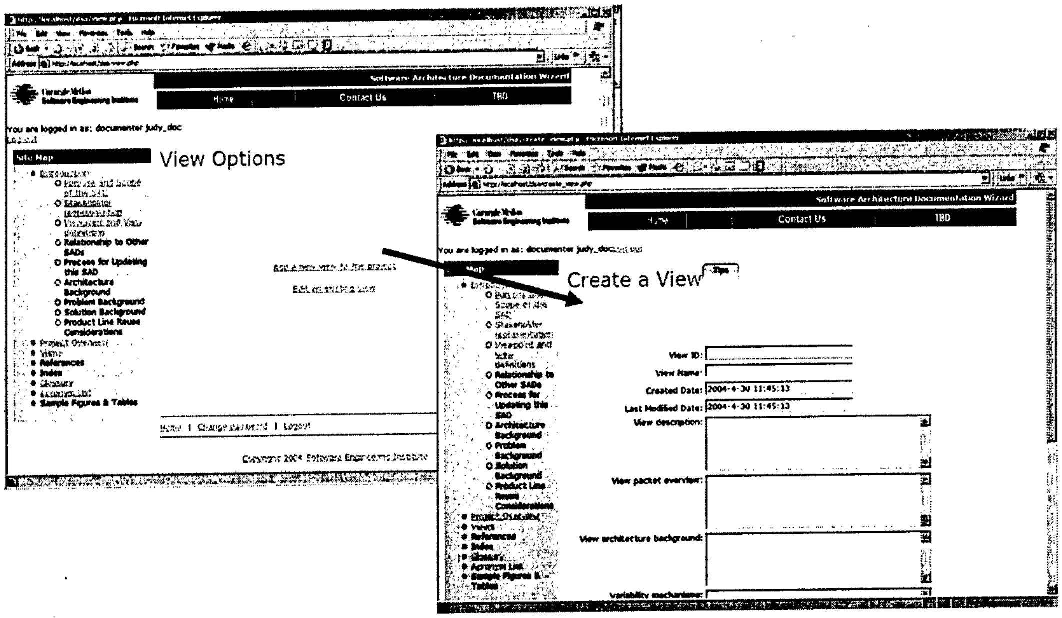


Figure 9: Documenter's View

The documenter may type directly into the form, cut and paste text from another source, or upload a document.

3.6 Tips

Tips (helpful information) are available on a role-by-role basis. Stakeholders have different needs for support in understanding and creating documentation. The system meets these needs by making relevant information available to users based on both the level of support they expect and the type of support appropriate to their role. Figure 10 displays the tip available to a developer from the Views screen that describes a view.

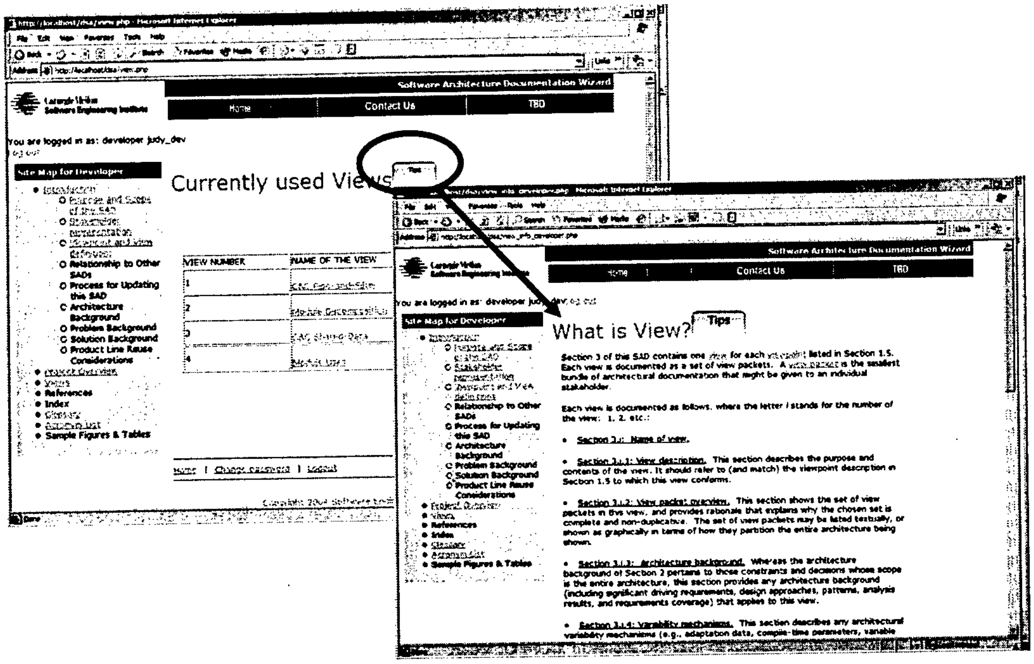


Figure 10: View Tip Provided for a Developer

The tip accessed by a documenter, shown in Figure 11, describes what to include when documenting a view.

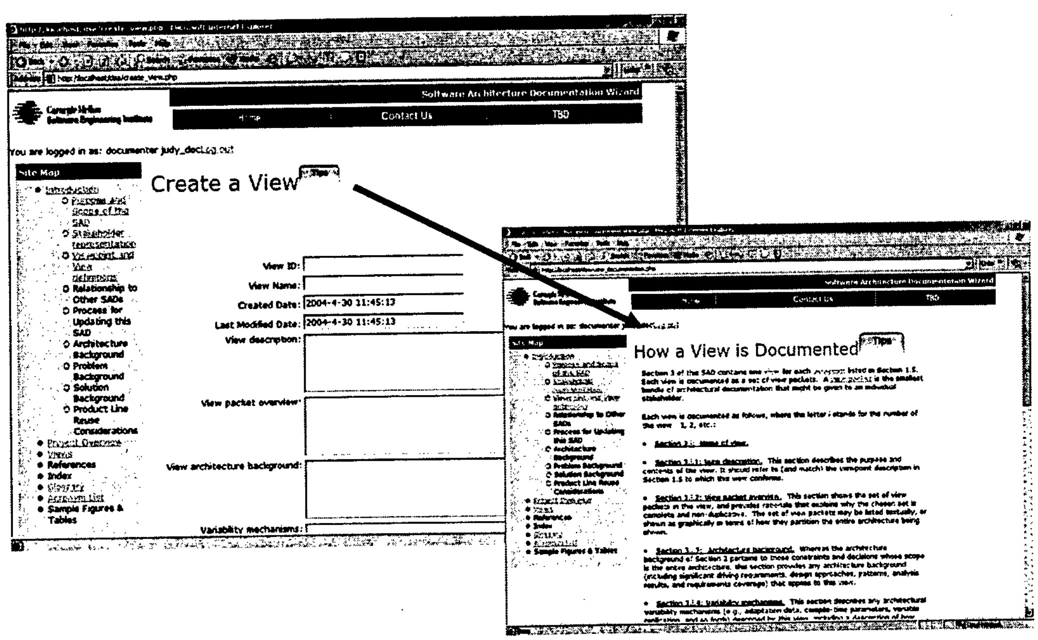


Figure 11: View Tip Provided for a Documenter

3.7 Flash Diagrams as Mini-Tutorials

As described in Section 2.2.1, flash diagrams can serve as mini-tutorials within the tips environment or be hyperlinked from any relevant point in the documentation. Figure 12 shows a page from a tutorial describing the C&C shared-data view.

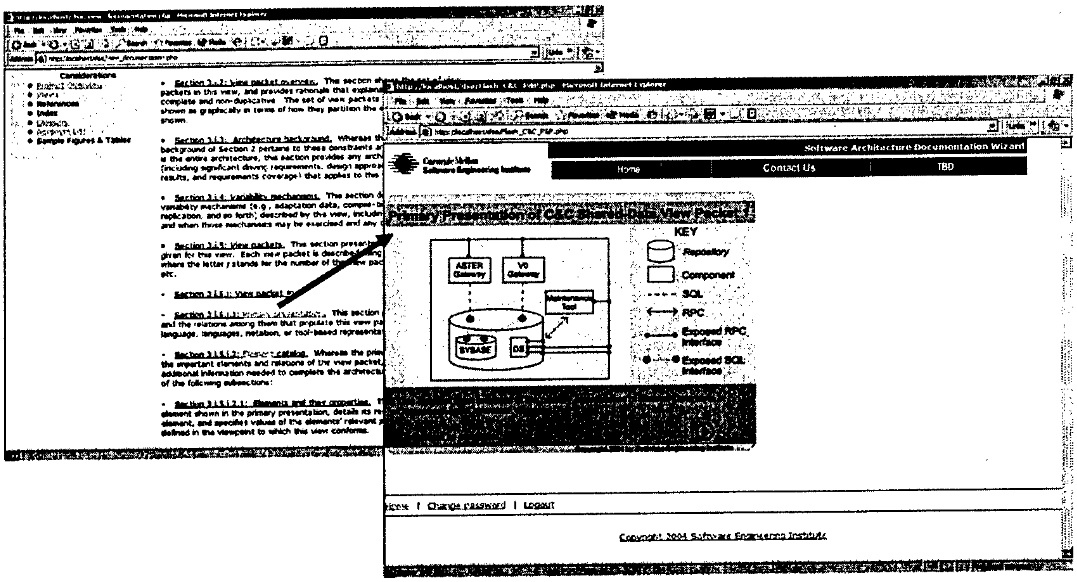


Figure 12: Mini-Tutorial Link Activated

4 Summary

The DSA Wizard project pursued the following goals:

- Create a template that provides a structure and guidance for documenting software architecture similar to that offered by the SAD Template, which supports documentation using Microsoft Word. This template also provides clickable documentation guidance and hyperlinking capability for increased accessibility to recorded information.
- Address the documentation needs of a wide audience of stakeholders in a distributed software environment. Very early in the design process, we recognized that stakeholders have different needs with regards to their perspective of the software and the documentation. Documentation-related needs vary according to the helpfulness of specific information and the location of hyperlinks. We endeavored to develop a system providing stakeholders with different document views and guidance to help them create and use the documentation.
- Make the document easily accessible around the globe.

Architecting a software system is an iterative and incremental process that must first describe the architecture in terms of its major elements. Then each element is gradually elaborated and refined as the architecture grows and evolves in response to reviews and discoveries during the development process. Therefore, software architecture documentation is best viewed as a living entity; it must mature gracefully with the software that it describes.

We take a Web-based, repository-backed approach to documentation, which is more appropriate to stakeholder needs and requirements for a living document than traditional documentation techniques. Our approach provides fine-grained access to document artifacts for multiple authors and readers in concert. The design of the DSA Documentation System endeavors to overcome the following specific problems inherent in creating and using paper and other traditional forms of documentation:

- Typically acronyms are described only the first time they are used; that meaning is often forgotten by the time readers encounter the next use of the acronym, requiring them to either trace back through the document for the meaning or find their way to the acronym list and back to their place in the document. This problem also occurs with the use of system- and organization-specific definitions and names and is very distracting to readers.
- The flat nature of traditional documentation inhibits the ability to provide guidance along with template structure. The original SAD Template provides guidance and other general information inline with the documentation of a specific system. Some of this information

is useful only to the document's author; some also is useful to readers. Thus, removing some text before releasing the document might be necessary, requiring decisions about what to remove and how to remove it so that the removal does not harm the document. This process becomes costly and error prone.

- Traditional documentation is presented in a single format; however, parts of a document might be better recorded and presented in several formats. For instance, the best way to describe a behavioral aspect of the software uses an active diagram that steps through the activity being represented.
- A traditional document, in the effort to reduce bulk and still address the needs of a general audience, inevitably contains much information that is unneeded by some users and lacks information required by others. The more sophisticated user skims the document, perhaps missing key elements, and other users become frustrated and search for different reference materials.
- Updates to the document must be distributed and incorporated into documents held by numerous users, requiring careful records as to who has copies. And, the distribution of revisions does not ensure that users incorporate them, inevitably leading to stakeholders referencing inconsistent documents.
- Documentation created for a general audience suffers from its sheer size. Notebooks full of paper are cumbersome, and large digital files are difficult to navigate and search.
- Users of documentation are the best sources of information about how to improve documentation, but, generally speaking, no simple mechanism exists for communicating suggestions back to the document's author.

Powerful tools exist today to help provide documentation that does not suffer from these problems. The characteristics of the Web and power of repositories combine to provide an environment that supports both the form and the spirit of the DSA approach to software architecture documentation. Using these tools, we have addressed each issue listed above in the conceptual design for the DSA Documentation System:

- An acronym list and glossary are required elements of any document and must be hyperlinked both internally and from within the document.
- Specific aspects of the DSA Documentation System are included specifically to provide general information and guidance in role-specific forms, thus eliminating problems associated with information removal that might be required when using the SAD Template.
- Various aspects of documentation can be captured and displayed in formats that are most appropriate to the information and the context of its current use.
- The needs of each stakeholder are addressed through customized views automatically generated based on the role configured by the architect or other authorized person.

- Revisions are available immediately or when deemed appropriate by the architect on a role-by-role basis, and, because the information is stored and accessed from a central repository, different users of the same role type will always access the same information.
- Due to the nature of the online medium, size is not an issue for architecture documentation captured in a DSA Documentation System.
- Web-based environments offer avenues for providing feedback easily and quickly. For example, simple feedback links that raise an email composition window to the appropriate person can be included in all Web-page frames.

The DSA Wizard prototype demonstrates the power of this approach to documentation; however, the process of creating the wizard uncovered other issues associated with using such flexible technologies. The following questions remain:

- What constitutes a version when parts of the document are available in different forms to different users? How can the wide variety of possible configurations be managed?
- Are the costs associated with providing tailored views too high? A balance must be achieved between the additional complexity of using a highly configurable system and the advantage of providing stakeholder-appropriate views of the documentation.
- How can the entire documentation package be printed? It seems inevitable that some user will want to print the entire document. When the documentation package consists of a collection of small files in many formats, which may be traversed in a variety of ways, how are the structure and format of a printed document determined for a given user at a given time?
- Can security be adequately maintained?

As we explore content management systems, looking for candidate target platforms for the DSA Documentation System, we hope to answer some of these questions but know that in doing so we are likely to raise additional ones. When faced with these questions, we will continue to explore solutions in order to support the creation and use of software architecture documentation in a stakeholder-centric manner.

5 Conclusions and Future Directions

Creating software architecture documentation that is usable by distributed teams and that remains consistent and current is best accomplished in a Web-based environment. The design for a documentation system based on the SAD template (a Microsoft Word template) includes additional features to support role-based access at a fine level of granularity and provides different views of the document on a role-by-role basis. Such a design, therefore, supports stakeholders' needs in a personalized way and increases the likelihood that the document will be used and remain a central artifact of the development process throughout the life of the software system.

The DSA Wizard prototype presented in Section 3 is a standalone version of the DSA Documentation System design. We are currently exploring the possibility of including the design as a template in a commercial content management system, which would enhance configuration management support and provide a robust environment in which to create and use software architecture documentation.

We are taking an iterative approach to the DSA Documentation System design. The exploration of commercial tool support for that system is likely to spark ideas for new features and might also expose limitations imposed by current technology for other features; the design will be adapted appropriately.

References

[Clements 03] Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley, 2003.

Appendix A: SAD Microsoft Word Template

<Insert OrganizationName>

<Insert Name of SAD>
Software Architecture Document
(SAD)

CONTENT OWNER: <Insert Name>

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
•	•	•

TIPS FOR USING THIS TEMPLATE

To turn on helpful toolbars:

Use the *DID Styles* toolbar to apply the Word Styles that will be used often in this template.
Displaying this toolbar will provide quick access to commonly used DID Styles.
(If the DID Styles Toolbar is not showing, turn it on by selecting *View > Toolbars > DID Styles*.)

To create an instance of this document:

- Insert relevant information on cover sheet.
- Insert relevant information in page header: Move to a page of the body of the report, select *View > Header and Footer* from the main menu, and then replace relevant information in the header box at the top of the page.
- A gray box containing *CONTENT OF THIS SECTION* is provided at the beginning of most sections and subsections. After determining what specific information will be included in your document, you can remove this gray box or leave it to serve as a section overview for your readers. In the case that text has been provided in the template, the provided text should be inspected for relevance and revised as necessary. In all cases, relevant text should be added to each section or subsection as appropriate for your document.
- Consider hyperlinking key words used in the document with their entries in the Glossary or other location in which they are defined.

To update the contents and page numbers in the Table of Contents, List of Figures, and List of Tables:

- Position the cursor anywhere in the Table of Contents.
- Click the *F9* function key.
- Answer "Update entire table" to each of the next three windows.

To insert a figure caption:

- From the main menu, choose *Insert > Caption > Figure*.
- Click the OK button.
- Add a colon and a tab stop after the figure number in the caption itself.

The figure caption should use the *Caption* style.

To insert a table caption:

- From the main menu, choose *Insert > Reference > Caption > Table*.
- Click the OK button.
- Then add a colon and a tab stop after the table number in the caption itself.

The table caption should use the *Caption* style.

ORGANIZING THE DOCUMENTATION INTO VOLUMES

Consider packaging your SAD as a multi-volume set of documentation.

It is often helpful to break your documentation into more than one volume so that the document does not become unwieldy. There are many ways that this can be accomplished. The structuring of the document must support the needs of the intended audience and must be determined in the context of the project. Each document that you produce should include the date of issue and status; draft, baseline, version number, name of issuing organization; change history; and a summary. A few decomposition options are:

- *A 2-Volume approach:* Separate the documentation into two volumes; one that contains the views of the software architecture and one that contains everything else. A common variant of this approach has one volume per view, and one volume for everything else.
- *A 3-Volume approach:* Document organizational policies, procedures, and the directory in one volume, system-specific overview material in a second, and view documentation in a third.
- *A 4-Volume approach:* Create one volume for each viewtype [Clements 03] (module, component-and-connector, allocation) that contains the documentation for the relevant views. Include all of the other information in the fourth volume.
- Software interfaces are often documented in a separate volume.

In any case, the readers should begin with the volume containing the Documentation Roadmap (Section 1 in this template).

Contents

<Insert OrganizationName>

List of Figures

<Insert OrganizationName>

<Insert OrganizationName>

List of Tables

<Insert OrganizationName>

<Insert OrganizationName>

1 Documentation Roadmap and Overview

Subparts of Section 1 provide information that will help readers or users of the Software Architecture Document (SAD) quickly find information that will enable them to do their jobs. Readers of the SAD seeking an overview should begin here, as should readers interested in finding particular information to answer a specific question. **See especially Section 1.3** (which explains the information that is found in each section of the SAD), **Section 1.4** (which explains the stakeholders for which the SAD has been particularly aimed), **Section 1.5** (which explains the viewpoints supported in this SAD), **and Section 1.6** (which explains how architectural views are documented in this SAD).

1.1 Document Management and Configuration Control Information

CONTENTS OF THIS SECTION: This section identifies the version, release date, and other relevant management and configuration control information associated with the current version of the document. Optional items for this section include: change history and indication of significant changes from version to version.

- Revision Number: << >>
- Revision Date: << >>
- Purpose of Revision: << >>
- Scope of Revision: <<*list sections or page numbers that have been revised*>>

1.2 Purpose and Scope of the SAD

CONTENTS OF THIS SECTION: This section explains the SAD's overall purpose and scope, the criteria for deciding which design decisions are architectural (and therefore documented in the SAD), and which design decisions are non-architectural (and therefore documented elsewhere).

This SAD specifies the software architecture for **<insert scope of SAD>**. All information regarding the software architecture may be found in this document, although much information is incorporated by reference to other documents.

The software architecture for a system[1] is the structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 03]. "Externally visible properties" refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on. This definition provides the basic litmus test for what information is included in this SAD, and what information is relegated to downstream documentation.

[1] Here, a system may refer to a system of systems.

The software architecture first and foremost embodies information about how the elements relate to each other. This means that architecture specifically omits certain information about elements that does not pertain to their interaction. Thus, a software architecture is an *abstraction* of a system that suppresses details of elements that do not affect how they use, are used by, relate to, or interact with other elements. Elements interact with each other by means of interfaces that partition details about an element into public and private parts. Software architecture is concerned with the public side of this division, and that will be documented in this SAD accordingly. On the other hand, private details of elements—details having to do solely with internal implementation—are not architectural and will not be documented in a SAD.

The definition of software architecture makes it clear that systems can and do comprise more than one structure and that no one structure holds the irrefutable claim to being the architecture. The neurologist, the orthopedist, the hematologist, and the dermatologist all take a different perspective on the structure of a human body. Ophthalmologists, cardiologists, and podiatrists concentrate on subsystems. And the kinesiologist and psychiatrist are concerned with different aspects of the entire arrangement's behavior. Although these perspectives are pictured differently and have very different properties, all are inherently related; together they describe the architecture of the human body. So it is with software. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures. To communicate meaningfully about an architecture, we must make clear which structure or structures we are discussing at the moment—which *view* we are taking of the architecture. Thus, this SAD follows the principle that documenting a software architecture is a matter of documenting the relevant views and then documenting information that applies to more than one view.

For example, all non-trivial software systems are partitioned into implementation units; these units are given specific responsibilities and are the basis of work assignments for programming teams. This kind of element will comprise programs and data that software in other implementation units can call or access, and programs and data that are private. In large projects, the elements will almost certainly be subdivided for assignment to subteams. This is one kind of structure often used to describe a system. It is a very static structure, in that it focuses on the way the system's functionality is divided up and assigned to implementation teams.

Other structures are much more focused on the way the elements interact with each other at runtime to carry out the system's function. Suppose the system is to be built as a set of parallel processes. The set of processes that will exist at runtime, the programs in the various implementation units that were described previously and are strung together sequentially to form each process, and the synchronization relations among the processes form another kind of structure often used to describe a system.

None of these structures alone is *the* architecture, although they all convey architectural information. The architecture consists of these structures as well as many others. This example shows that

<Insert OrganizationName>

since architecture can comprise more than one kind of structure, there is more than one kind of element (e.g., implementation unit and processes), more than one kind of interaction among elements (e.g., subdivision and synchronization), and even more than one context (e.g., development time versus runtime). By intention, the definition does not specify what the architectural elements and relationships are. Is a software element an object? A process? A library? A database? A commercial product? It can be any of these things and more.

Although software architecture tends to focus on structural information, the *behavior of each element is part of the software architecture* insofar as that behavior can be observed or discerned from the point of view of another element. This behavior is what allows elements to interact with each other, which is clearly part of the software architecture and will be documented in the SAD as such. Behavior is documented in the element catalog of each view.

1.3 How the SAD Is Organized

CONTENTS OF THIS SECTION: This section provides a narrative description of the seven major sections of the SAD and the overall contents of each. Readers seeking specific information can use Section 1.3 to help them locate it more quickly.

This SAD is organized into the following seven sections:

- **Section 1 provides information about this document and its intended audience.** It provides the roadmap and document overview. Every reader who wishes to find information relevant to the software architecture described in this document should begin by reading Section 1, which describes how the document is organized, which stakeholder viewpoints are represented, how stakeholders are expected to use the document, and where information may be found. Section 1 also provides information about the viewpoints that are used by this SAD to communicate the software architecture.
- **Section 2 provides information about the software architecture.** It describes the background and rationale for the software architecture. It explains the constraints and influences that led to the current architecture and describes the major architectural approaches that have been utilized in the architecture.
- **Sections 3 and 4 specify the software architecture.**
- **Sections 5, 6, and 7 provide reference information.** Section 5 provides look-up information for documents that are cited elsewhere in this SAD. Section 6 is a *directory*, which is an index of architectural elements and relations telling where each one is defined in this SAD and where each is used in this SAD. Section 7 provides a glossary and acronym list.

1.4 Stakeholder Representation

CONTENTS OF THIS SECTION: This section provides a list of the stakeholder roles considered in the development of the architecture described by this SAD. For each, the section lists the concerns that the stakeholder has that can be addressed by the information in this SAD. This information is represented as a matrix, where the rows list stakeholder roles, the columns list concerns, and a cell in the matrix contains an indication of how serious the concern is to a stakeholder in that role. This information is used to motivate the choice of viewpoints chosen in Section 1.5. The list below provides a sample set of stakeholders for a large organization that may be developing a complex system. For example, a maintainer's concerns include knowing what software elements may be affected by a change. This list will be tailored for each organization that is developing a SAD.

ANSI/IEEE 1471-2000 requires that at least the following stakeholders be considered: users, acquirers, developers, and maintainers.

• Customer • Application software developers • Infrastructure software developers • End users • Application system engineers • Application hardware engineers	• Project manager • Communications engineers • Chief engineer/chief scientist • Program managers • System and software integration and test engineers • Safety engineers and certifiers	• External organizations • Operational system managers • Trainers • Maintainers • Auditors • Security engineers and certifiers

1.5 Viewpoint Definitions

CONTENTS OF THIS SECTION: This section provides a short textual definition of a viewpoint and how the concept is used in this SAD. The section describes viewpoints that may be used in the SAD. The specific viewpoints will be tailored by the organization.

The SAD employs a stakeholder-focused, multiple-view approach to architecture documentation, as required by ANSI/IEEE 1471-2000, the recommended best practice for documenting the architecture of software-intensive systems [IEEE 00].

Each stakeholder of a software system—customer, user, project manager, coder, analyst, tester, and so on—is concerned with different characteristics of the system that are affected by its software architecture. For example, the user is concerned that the system is reliable and available when needed; the customer is concerned that the architecture can be implemented on schedule and within budget; the manager is worried (in addition to cost and schedule) that the architecture will allow teams to work largely independently, interacting in disciplined and controlled ways. The developer is worried about strategies to achieve all of those goals. The security analyst is concerned that the system will meet its information assurance requirements, and the performance analyst is similarly concerned with it satisfying real-time deadlines.

As described in Section 1.2, a software architecture comprises more than one software structure, each of which provides an engineering handle on different system qualities. A *view* is the specification of one or more of these structures, and documenting a software architecture, then, is a mat-

<Insert OrganizationName>

ter of documenting the relevant views and then documenting information that applies to more than one view [Clements 03].

ANSI/IEEE 1471-2000 provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. A viewpoint identifies the set of concerns to be addressed and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view. A view, then, is a viewpoint applied to a system. It is a representation of a set of software elements, their properties, and the relationships among them that conform to a defining viewpoint. Together, the chosen set of views show the entire architecture and all of its relevant properties. A SAD contains the viewpoints, relevant views, and information that applies to more than one view to give a holistic description of the system.

Architectural views can, by and large, be divided into three groups, depending on the broad nature of the elements they show:

- *Module views*. Here, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility and are assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Module structures allow us to answer questions such as: What is the primary functional responsibility assigned to each module? What other software elements is a module allowed to use? What other software does it actually use? What modules are related to other modules by generalization or specialization (i.e., inheritance) relationships?
- *Component-and-connector views*. Here, the elements are runtime components (which are principal units of computation) and connectors (which are the communication vehicles among components). Component and connector structures help answer questions such as: What are the major executing components and how do they interact? What are the major shared data stores? Which parts of the system are replicated? How does data progress through the system? What parts of the system can run in parallel? How can the system's structure change as it executes?
- *Allocation views*. These views show the relationship between the software elements and elements in one or more external environments in which the software is created and executed. Allocation structures answer questions such as: What processor does each software element execute on? In what files is each element stored during development, testing, and system building? What is the assignment of the software element to development teams?

These three kinds of structures correspond to the three broad kinds of decisions that architectural design involves:

- How is the system to be structured as a set of code units (modules)?
- How is the system to be structured as a set of elements that have runtime behavior (components) and interactions (connectors)?
- How is the system to relate to non-software structures in its environment (such as CPUs, file systems, networks, development teams, etc.)?

The remainder of Section 1.5 defines the viewpoints used in this SAD. The following table summarizes the stakeholders in this project and the viewpoints that have been included to address their concerns.

Table 1: Stakeholders and Relevant Viewpoints

Stakeholder	Viewpoints

1.5.1 <Insert name of viewpoint> Viewpoint

Example viewpoint definition:

Viewpoint #1. **Module decomposition viewpoint**

1.5.1.1 Abstract. Views conforming to the module decomposition viewpoint partition the system into a unique non-overlapping set of hierarchically decomposable implementation units (*modules*).

1.5.1.2 Stakeholders and Their Concerns Addressed. Stakeholders and their concerns addressed by this viewpoint include

- project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules;
- COTS specialists, who need to have software elements defined as units of functionality, so they can search the marketplace and perform trade studies to find suitable COTS candidates;
- testers and integrators who use the modules as their unit of work;
- configuration management specialists who are in charge of maintaining current and past versions of the elements;
- system build engineers who use the elements to produce a running version of the system;
- maintainers, who are tasked with modifying the software elements;
- implementers, who are required to implement the elements;
- software architects for those software elements sufficiently large or complex enough to warrant their own software architectures;
- the customer, who is concerned that projected changes to the system over its lifetime can be made economically by confining the effects of each change to a small number of elements.

1.5.1.3 Elements, Relations, Properties, and Constraints. Elements of the module decomposition viewpoint are modules, which are units of implementation that provide defined functionality. Modules are hierarchically decomposable; hence, the relation is "is-part-of." Properties of elements include their names, the functionality assigned to them (including a statement of the quality attributes associated with that functionality), and their software-to-software interfaces. The module properties may include requirements allocation, supporting requirements traceability.

1.5.1.4 Language(s) to Model/Represent Conforming Views. Views conforming to the module decomposition viewpoint may be represented by (a) plain text using indentation or outline form [Clements 03]; (b) UML, using subsystems or classes to represent elements and "is-part-of" or nesting to represent the decomposition relation.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria. Completeness/consistency criteria include (a) no element has more than one parent; (b) major functionality is provided for

by exactly one element; (c) the union of all elements' functionality covers the requirements for the system; (d) every piece of source code can be mapped to an element in the module decomposition view (if not, the view is not complete); (e) the selection of module aligns with current and proposed procurement decisions. Additional consistency/completeness criteria apply to the specifications of the elements' interfaces. Applicable evaluation/analysis techniques include (a) scenario-based evaluation techniques such as ATAM [Clements 01] to assure that projected changes are supported economically by the decomposition; (b) disciplined and detailed mapping to requirements to assure coverage and non-overlapping functionality; (c) cost-based techniques that determine the number and composition of modules for efficient procurement.

1.5.1.6 Viewpoint Source. [Clements 03, Section 2.1] describes the module decomposition style, which corresponds in large measure to this viewpoint.

1.5.1.1 Abstract

CONTENTS OF THIS SECTION: In this section, a brief overview of the viewpoint is provided.

1.5.1.2 Stakeholders and Their Concerns Addressed

CONTENTS OF THIS SECTION: This section describes the stakeholders and their concerns that this viewpoint is intended to address. Listed are questions that can be answered by consulting views that conform to this viewpoint. Optionally, the section includes significant questions that cannot be answered by consulting views conforming to this viewpoint.

1.5.1.3 Elements, Relations, Properties, and Constraints

CONTENTS OF THIS SECTION: This section defines the types of elements, the relations among them, the significant properties they exhibit, and the constraints they obey for views conforming to this viewpoint.

1.5.1.4 Language(s) to Model/Represent Conforming Views

CONTENTS OF THIS SECTION: This section lists the language or languages that will be used to model or represent views conforming to this viewpoint, and cites, a definition document for each.

1.5.1.5 Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria

CONTENTS OF THIS SECTION: This section describes rules for consistency and completeness that apply to views in this viewpoint, as well as any analysis of evaluation techniques that apply to the view that can be used to predict qualities of the system whose architecture is being specified.

1.5.1.6 Viewpoint Source

CONTENTS OF THIS SECTION: This section provides a citation for the source of this viewpoint definition, if any.

1.6 How a View Is Documented

CONTENTS OF THIS SECTION: This section describes how the documentation for a view is structured and organized.

Section 3 of this SAD contains one view for each viewpoint listed in Section 1.5. Each view is documented as a set of view packets. A view packet is the smallest bundle of architectural documentation that might be given to an individual stakeholder.

Each view is documented as follows, where the letter *i* stands for the number of the view: 1, 2, etc.:

- Section 3.i: Name of view.
- Section 3.i.1: View description. This section describes the purpose and contents of the view. It should refer to (and match) the viewpoint description in Section 1.5 to which this view conforms.
- Section 3.i.2: View packet overview. This section shows the set of view packets in this view and provides rationale that explains why the chosen set is complete and non-duplicative. The set of view packets may be listed textually or shown graphically in terms of how they partition the entire architecture being shown.
- Section 3.i.3: Architecture background. Whereas the architecture background of Section 2 pertains to those constraints and decisions whose scope is the entire architecture, this section provides any architecture background (including significant driving requirements, design approaches, patterns, analysis results, and requirements coverage) that applies to this view.
- Section 3.i.4: Variability mechanisms. This section describes any architectural variability mechanisms (e.g., adaptation data, compile-time parameters, variable replication, and so forth) described by this view, including a description of how and when those mechanisms may be exercised and any constraints on their use.
- Section 3.i.5: View packets. This section presents all of the view packets given for this view. Each view packet is described using the following outline, where the letter *j* stands for the number of the view packet being described: 1, 2, etc.
  - Section 3.i.5.j: View packet #j.
  - Section 3.i.5.j.1: Primary presentation. This section presents the elements and the relations among them that populate this view packet, using an appropriate language, languages, notation, or tool-based representation.
  - Section 3.i.5.j.2: Element catalog. Whereas the primary presentation shows the important elements and relations of the view packet, this section provides additional information needed to complete the architectural picture. It consists of the following subsections:
    - Section 3.i.5.j.2.1: Elements. This section describes each element shown in the primary presentation, details the responsibilities of each element, and specifies values of the elements' relevant *properties*, which are defined in the viewpoint to which this view conforms.
    - Section 3.i.5.j.2.2: Relations. This section describes any additional relations among elements shown in the primary presentation, or specializations or restrictions on the relations shown in the primary presentation.

  - Section 3.i.5.j.2.3: Interfaces. This section specifies the software interfaces to any elements shown in the primary presentation that must be visible to other elements.
  - Section 3.i.5.j.2.4: Behavior. This section specifies any significant behavior of elements or groups of interacting elements shown in the primary presentation.
  - Section 3.i.5.j.2.5: Constraints: This section lists any constraints on elements or relations not otherwise described.
- Section 3.i.5.j.3: Context diagram. This section provides a context diagram showing the context of the part of the system represented by this view packet. It also designates the view packet's scope with a distinguished symbol and shows interactions with external entities in the vocabulary of the view.
- Section 3.i.5.j.4: Variability mechanisms. This section describes any variabilities that are available in the portion of the system shown in the view packet, along with how and when those mechanisms may be exercised.
- Section 3.i.5.j.5: Architecture background. This section provides rationale for any significant design decisions whose scope is limited to this view packet.
- Section 3.i.5.j.6: Relation to other view packets. This section provides references for related view packets, including the parent, children, and siblings of this view packet. Related view packets may be in the same view or in different views.

1.7 Relationship to Other SADs

CONTENTS OF THIS SECTION: This section describes the relationship between this SAD and other architecture documents, both system and software.

1.8 Process for Updating This SAD

CONTENTS OF THIS SECTION: This section describes the process a reader should follow to report discrepancies, errors, inconsistencies, or omissions from this SAD. The section also includes necessary contact information for submitting the report. If a form is required, either a copy of the blank form that may be photocopied is included or a reference to an online version is provided. This section also describes how error reports are handled, and how and when a submitter will be notified of the issue's disposition.

2 Architecture Background

2.1 Problem Background

CONTENTS OF THIS SECTION: The subparts of Section 2.1 explain the constraints that provided the significant influence over the architecture.

2.1.1 System Overview

CONTENTS OF THIS SECTION: This section describes the general function and purpose for the system or subsystem whose architecture is described in this SAD.

2.1.2 Goals and Context

CONTENTS OF THIS SECTION: This section describes the goals and major contextual factors for the software architecture. The section includes a description of the role software architecture plays in the life cycle, the relationship to system engineering results and artifacts, and any other relevant factors.

2.1.3 Significant Driving Requirements

CONTENTS OF THIS SECTION: This section describes behavioral and quality attribute requirements (original or derived) that shaped the software architecture. Included are any scenarios that express driving behavioral and quality attribute goals, such as those crafted during an SEI Quality Attribute Workshop (QAW)[2] [Barbacci 03] or software architecture evaluation using the Architecture Tradeoff Analysis Method® (ATAM®) [Bass 03].

2.2 Solution Background

CONTENTS OF THIS SECTION: The subparts of Section 2.2 provide a description of why the architecture is the way that it is, and a convincing argument that the architecture is the right one to satisfy the behavioral and quality attribute goals levied upon it.

---

2 Quality Attribute Workshop (QAW) is a product of the Carnegie Mellon® Software Engineering Institute (SEI).

® Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

<Insert OrganizationName>

2.2.1 Architectural Approaches

CONTENTS OF THIS SECTION: This section provides a rationale for the major design decisions embodied by the software architecture. It describes any design approaches applied to the software architecture, including the use of architectural styles or design patterns, when the scope of those approaches transcends any single architectural view. The section also provides a rationale for the selection of those approaches and describes any significant alternatives that were seriously considered and why they were ultimately rejected. The section describes any relevant COTS issues, including any associated trade studies.

2.2.2 Analysis Results

CONTENTS OF THIS SECTION: This section describes the results of any completed quantitative or qualitative analyses that provide evidence that the software architecture is fit for the purpose. If an ATAM evaluation has been performed, the results are included in the analysis sections of this final report. This section refers to the results of any other relevant trade studies, quantitative modeling, or other analysis results.

2.2.3 Requirements Coverage

CONTENTS OF THIS SECTION: This section describes the requirements (original or derived) addressed by the software architecture, with a short statement about where in the architecture each requirement is addressed.

2.2.4 Summary of Changes in Current Version

CONTENTS OF THIS SECTION: For versions of the SAD after the original release, this section summarizes the actions, decisions, decision drivers, requirements changes and analysis and trade study results that became decision drivers, and how these decisions have caused the architecture to evolve or change.

2.3 Product Line Reuse Considerations

CONTENTS OF THIS SECTION: When a product line is being developed, this section details how the software covered by this SAD is planned or expected to be reused in order to support the product line vision. In particular, this section includes a complete list of the variations that are planned to be produced and supported. "Variation" refers to a variant of the software produced through the use of pre-planned variation mechanisms made available in the software architecture. It may refer to a variant of one of the modules identified in this SAD, or a collection of modules, or the entire system or subsystem covered by this SAD. For each variation, the section identifies the increment(s) of the software build in which (a) the variation will be available and (b) the variation will be used. Finally, this section describes any additional potential that exists to reuse one or more of the modules or their identified variations, even if this reuse is not currently planned for any increment.

<Insert OrganizationName>

3 Views

CONTENTS OF THIS SECTION: The subparts of Section 3 specify the views corresponding to the viewpoints listed in Section 1.5.

3.1 <Insert view name> View

CONTENTS OF THIS SECTION: For each view documented in this SAD, the subparts of Section 3.1 specify it using the outline given in Section 1.6.

3.1.1 View Description

3.1.2 View Packet Overview

3.1.3 Architecture Background

3.1.4 Variability Mechanisms

3.1.5 View Packets

CONTENTS OF THIS SECTION: This section describes each view packet in the view using the outline given in Section 1.6.

<Insert OrganizationName>

3.1.5.1 View packet # j

3.1.5.1.1 Primary Presentation

3.1.5.1.2 Element Catalog

3.1.5.1.2.1 Elements

3.1.5.1.2.2 Relations

3.1.5.1.2.3 Interfaces

3.1.5.1.2.4 Behavior

3.1.5.1.2.5 Constraints

3.1.5.1.3 Context Diagram

3.1.5.1.4 Variability Mechanisms

3.1.5.1.5 Architecture Background

3.1.5.1.6 Related View Packets

4 Relations Among Views

Each of the views specified in Section 3 provides a different perspective and design handle on a system, and each is valid and useful in its own right. Although the views give different system perspectives, they are not independent. Elements of one view will be related to elements of other views, and we need to reason about these relations. For example, a module in a decomposition view may be manifested as one, part of one, or several components in one of the component-and-connector views, reflecting its runtime alter-ego. In general, mappings between views are many to many. Section 4 describes the relations that exist among the views given in Section 3. As required by ANSI/IEEE 1471-2000, it also describes any known inconsistencies among the views.

4.1 General Relations Among Views

CONTENTS OF THIS SECTION: This section describes the general relationship among the views chosen to represent the architecture. Also in this section, consistency among those views is discussed and any known inconsistencies are identified.

4.2 View-To-View Relations

CONTENTS OF THIS SECTION: For each related set of views, this section shows how the elements in one view are related to elements in another.

<Insert OrganizationName>

5 Referenced Materials

CONTENTS OF THIS SECTION: This section provides citations for each reference document. Enough information is provided so that a reader of the SAD can be reasonably expected to locate the document.

Barbacci 03	Barbacci, M.; Ellison, R.; Lattanze, A.; Stafford, J.; Weinstock, C.; & Wood, W. *Quality Attribute Workshops (QAWs), Third Edition* (CMU/SEI-2003-TR-016, ADA418428). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2003. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr016.html>.
Bass 03	Bass, L; Clements, P.; & Kazman, R. *Software Architecture in Practice,* Second Edition. Boston, MA: Addison-Wesley Longman, 2003.
Clements 01	Clements, P.; Kazman, R; & Klein, N. *Evaluating Software Architectures: Methods and Case Studies.* Boston, MA: Addison-Wesley Longman, 2001.
Clements 03	Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley Longman, 2003.
IEEE 00	Institute of Electrical and Electronics Engineers, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems* (IEEE Std. 1471-2000). New York, NY: Institute of Electrical and Electronics Engineers, 2000.

<Insert OrganizationName>

6 Directory

6.1 Index

CONTENTS OF THIS SECTION: This section provides an index of all element names, relation names, and property names. For each entry, the following are identified:

- the location in the SAD where it was defined
- each place it was used

6.2 Glossary

CONTENTS OF THIS SECTION: This section provides a list of definitions of special terms and acronyms used in the SAD. If terms are used in the SAD that are also used in a parent SAD and the definition is different, this section explains why.

Term	Definition
software architecture	The structure or structures of that system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [Bass 03]. "Externally visible properties" refer to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
view	A representation of a whole system from the perspective of a related set of concerns [IEEE 00]. A representation of a particular type of software architectural elements that occur in a system, their properties, and the relations among them. A view conforms to a defining viewpoint.
view packet	The smallest package of architectural documentation that could be useful to a stakeholder. The documentation of a view is composed of

	one or more view packets.
viewpoint	A specification of the conventions for constructing and using a view; a pattern or template from which to develop individual views by establishing the purposes and audience for a view, and the techniques for its creation and analysis [IEEE 00]. Identifies the set of concerns to be addressed, and identifies the modeling techniques, evaluation techniques, consistency checking techniques, etc., used by any conforming view.

6.3 Acronym List

API	Application Programming Interface; Application Program Interface; Application Programmer Interface
ATAM	Architecture Tradeoff Analysis Method
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off-the-Shelf
EPIC	Evolutionary Process for Integrating COTS-Based Systems
IEEE	Institute of Electrical and Electronics Engineers
KPA	Key Process Area
OO	Object Oriented
ORB	Object Request Broker
OS	Operating System
QAW	Quality Attribute Workshop
RUP	Rational Unified Process
SDE	Software Development Environment
SEE	Software Engineering Environment
SEI	Software Engineering Institute Systems Engineering & Integration Software End Item
SEPG	Software Engineering Process Group
SLOC	Source Lines of Code

<Insert OrganizationName>

SW-CMM	Capability Maturity Model for Software
CMMI-SW	Capability Maturity Model Integration for Software Engineering
UML	Unified Modeling Language

7 Sample Figures & Tables

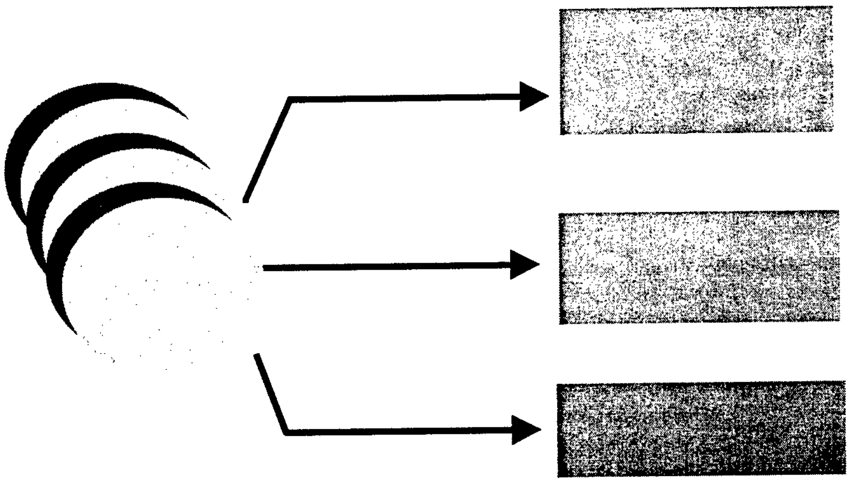


Figure 1: Sample Figure

Table 2: Sample Table

Table Heading	Table Heading	Table Heading	Table Heading
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body
Table Body	Table Body	Table Body	Table Body

<Insert OrganizationName>

Appendix A Appendices

CONTENTS OF THIS SECTION: Appendices may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data, API specification). As applicable, each appendix is referenced in the main body of the document where the data would normally have been provided. Appendices may be bound as separate documents for ease in handling.

A.1 Heading 2 - Appendix

A.2 Heading 2 - Appendix

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 2004	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Creating and Using Software Architecture Documentation Using Web-Based Tool Support		5. FUNDING NUMBERS F19628-00-C-0003
6. AUTHOR(S) Judith A. Stafford		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2004-TN-037
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Documenting software architecture (DSA) is a crucial facet in the development of a software system, yet often it is carried out in a haphazard fashion, if at all. Lack of attention to the documentation results from insufficient guidance about what should be documented and when and how to capture the information so that system stakeholders find it useful. The book *Documenting Software Architectures: Views and Beyond* provides such guidance in the DSA approach, and this report describes the conceptual design for a documentation system based on that approach. A system is envisioned that enables the architect to capture architectural decisions and related artifacts as a living repository that can communicate information to stakeholders who might be both geographically and temporally distributed. The system must communicate in a way that allows each stakeholder quick and easy access to information relevant to the person's role in the software development process. This report describes a design prototype that demonstrates a Web-based approach to creating, communicating, and using software architecture throughout the life of the system.

14. SUBJECT TERMS documenting software architecture, DSA, capturing software architecture, documentation system			15. NUMBER OF PAGES 63
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL